Visual Basic: Windows Controls

Visual Studio 6.0

UpDown Control

See Also Example Properties Methods Events

An **UpDown** control has a pair of arrow buttons which the user can click to increment or decrement a value, such as a scroll position or a value in an associated control, known as a buddy control.

Syntax

UpDown

Remarks

To the user, an **UpDown** control and its buddy control often look like a single control. The buddy control can be any control that can be linked to the **UpDown** control through the **BuddyControl** property, and usually displays data, such as a **TextBox** control or a **CommandButton** control.

Note Lightweight windowless controls, such as the intrinsic label control, can't be used as a buddy control.

By setting the **AutoBuddy** property, the **UpDown** control automatically uses the previous control in the tab order as its buddy control. If there is no previous control in the tab order, the **UpDown** control will use the next control in the tab order as its buddy control. Another way to set the buddy control is with the **BuddyControl** property. At design time, when either the **AutoBuddy** property or the **BuddyControl** property is set, the buddy control will automatically pair up with the **UpDown** control by sizing and positioning next to it. The **UpDown** control can be positioned to the right or left of its buddy control with the **Alignment** property.

Note The **Updown** control relies on the Visual Basic implementation of extended properties like "Tab Index" and "Name." Because the VC++ resource editor does not support these properties, this control will not work with an MFC dialog app.

The **Increment**, **Min**, **Max**, and **Wrap** properties specify how the **UpDown** control's **Value** property changes when the user clicks the buttons on the control. For example, if you have values that are multiples of 10, and range from 20 to 80, you can set the **Increment**, **Min**, and **Max** properties to 10, 20, and 80, respectively. The **Wrap** property allows the **Value** property to increment past the **Max** property and start again at the **Min** property, or vice versa.

An **UpDown** control without a buddy control functions as a sort of simplified scroll bar.

Note The **UpDown** control should be used in place of the Spin Button control from Visual Basic 4.0.

Distribution Note The **UpDown** control is part of a group of ActiveX controls that are found in the MSCOMCT2.OCX file. To use the **UpDown** control in your application, you must add the MSCOMCT2.OCX file to the project. When distributing your application, install the MSCOMCT2.OCX file in the user's Microsoft Windows SYSTEM directory. For more information on how to add a custom control to a project, see the *Programmer's Guide*.

Visual Basic Reference

Visual Studio 6.0

UserControl Object

See Also Example Properties Methods Events

The **UserControl** object is the base object used to create an ActiveX control.

Remarks

An ActiveX control created with Visual Basic is always composed of a **UserControl** object, plus any controls referred to as constituent controls that you choose to place on the **UserControl**.

Like Visual Basic forms, **UserControl** objects have code modules and visual designers. Place constituent controls on the **UserControl** objects designer, just as you would place controls on a form.

Visual Basic Reference

Visual Studio 6.0

UserDocument Object

See Also Example Properties Methods Events

The base object of an ActiveX document, the **UserDocument** object resembles a standard Visual Basic **Form** object with some exceptions.

Remarks

The **UserDocument** object has most, but not all, of the events that are found on a **Form** object. The events present on a **Form** that are not found on the **UserDocument** include: Activate, Deactivate, LinkClose, LinkError, LinkExecute, LinkOpen, Load, QueryUnload, and Unload events.

Events present on the **UserDocument**, but not found on a **Form** object include: AsyncReadComplete, EnterFocus, ExitFocus, Hide, InitProperties, ReadProperties, Scroll, Show, and WriteProperties events.

You cannot place embedded objects (such as an Excel or Word document) or an **OLE Container** control on a **UserDocument**.

Visual Studio 6.0

Visual Basic: MSChart Control

ValueScale Object

See Also Example Properties Methods Events

Scale used to display a value axis.

Syntax

ValueScale

Visual Basic: MSChart Control

ValueScale Object Example

The following example sets the major and minor grid line color for a two-dimensional bar chart using the **ValueScale** object.

```
Private Sub Command1_Click()
   ' Set chart type to 2d bar.
   MSChart1.ChartType = VtChChartType2dBar
   ' Use manual scale to display y axis (value axis).
   With MSChart1.Plot.Axis(VtChAxisIdY).ValueScale
      .Auto = False
      .MajorDivision = 2
      .MinorDivision = 5
   End With
   ' Show major grid line in red and minor grid line
   ' in blue.
   With MSChart1.Plot.Axis(VtChAxisIdY).AxisGrid
      .MajorPen.VtColor.Set 255, 0, 0
      .MajorPen.Width = 4
      .MinorPen.VtColor.Set 0, 0, 255
      .MinorPen.Width = 2
   End With
End Sub
```

Visual Basic Extensibility Reference

Visual Studio 6.0

VBComponent Object

See Also Example Properties Methods Events Specifics

Represents a component, such as a class module or standard module, contained in a project.

Remarks

Use the **VBComponent** object to access the code module associated with a component or to change a component's property settings.

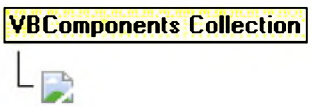
You can use the **Type** property to find out what type of component the **VBComponent** object refers to. Use the **Collection** property to find out what collection the component is in.

Visual Basic Extensibility Reference

Visual Studio 6.0

VBComponents Collection

See Also Example Properties Methods Events Specifics



Represents the components contained in a project.

Remarks

Use the **VBComponents** collection to access, add, or remove components in a project. A component can be a form, module, or class. The **VBComponents** collection is a standard collection that can be used in a **For Each** block.

You can use the **Parent** property to return the project the **VBComponents** collection is in.

In Visual Basic for Applications, you can use **Import** method to add a component to a project from a file.

Visual Basic Extensibility Reference

Visual Studio 6.0

VBComponentsEvents Object

See Also Example Properties Methods Events

Represents a source of events that occur when an object is added, removed, selected, renamed, or activated in a Visual Basic project.

Syntax

VBComponentsEvents

Visual Basic Extensibility Reference

Visual Studio 6.0

VBControl Object

See Also Example Properties Methods Events

Represents a control on a component in a project.

Syntax

VBControl

Remarks

A program can access a control through the **VBForm** object. Using the **VBForm** object, you can:

- Access all the design time properties of a control.
- Identify the container of the control.
- Change the Z-order of the control.

The **VBControl** object replaces the **ControlTemplate** object from Visual Basic version 4.0.

Visual Basic Reference

Visual Studio 6.0

VBControlExtender Object

See Also Example Properties Methods Events

Represents the Visual Basic VBControlExtender properties.

Syntax

VBControlExtender

Remarks

The **VBControlExtender** object is primarily used when dynamically adding a control to the **Controls** collection using the **Add** method. The **VBControlExtender** object is particularly useful for this purpose because it provides a generic set of properties, events, and methods to the developer. Another feature of the object is the ObjectEvent event which is designed to parse any event raised by a dynamically added control. The example below declares an object variable as **VBControlExtender**, and sets the variable when adding a control. The example also shows how you can program the ObjectEvent event.

```
Option Explicit
Dim WithEvents objExt As VBControlExtender ' Declare VBControlExtender variable WithEvents

Private Sub LoadControl()
   Licenses.Add "Project1.Control1", "ewrinvcmcoe"
   Set objExt = Controls.Add("Project1.Control1", "myCtl")
   objExt.Visible = True ' The control is invisible by default.
End Sub

Private Sub extObj_ObjectEvent(Info As EventInfo)
   ' Program the events of the control using Select Case.
   Select Case Info.Name
   Case "Click"
      ' Handle Click event here.
   ' Other cases now shown
   Case Else ' Unknown Event
      ' Handle unknown events here.
   End Select
End Sub
```

Restrictions on Setting the References to the Variable

There is one caveat to be aware of when setting the VBControlExtender object to a dynamically added control: intrinsic controls cannot be set to the variable.

Visual Basic Extensibility Reference

Visual Studio 6.0

VBControls Collection

See Also Example Properties Methods Events

Returns a collection all components on a form.

Syntax

VBControls

Remarks

A program can access controls through the **VBControls** collection. Using the **VBControls** collection, you can:

- Access all the controls on a component.
- Step through the collection of controls.
- Return a specific control.
- Add controls to a component.

The **Item** method determines the default value of the **VBControls** collection.

The **VBControls** collection replaces the **ControlTemplates** collection from Visual Basic version 4.0.

Visual Basic Extensibility Reference

Visual Studio 6.0

VBControlsEvents Object

See Also Example Properties Methods Events

Represents a source of events that occur when a control is added, removed, selected, renamed, or activated in a Visual Basic project.

Syntax

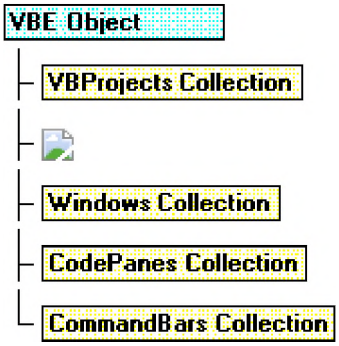
VBControlsEvents

Visual Basic Extensibility Reference

Visual Studio 6.0

VBE Object

See Also Example Properties Methods Events Specifics



The root object that contains all other objects and collections represented in Visual Basic for Applications.

Remarks

You can use the following collections to access the objects contained in the **VBE** object:

- Use the **VBProjects** collection to access the collection of projects.
- Use the **AddIns** collection to access the collection of add-ins.
- Use the **Windows** collection to access the collection of windows.
- Use the **CodePanes** collection to access the collection of code panes.
- Use the **CommandBars** collection to access the collection of command bars.

Use the **Events** object to access properties that enable add-ins to connect to all events in Visual Basic for Applications. The properties of the **Events** object return objects of the same type as the property name. For example, the **CommandBarEvents** property returns the **CommandBarEvents** object.

You can use the **SelectedVBComponent** property to return the active component. The active component is the component that is being tracked in the Project window. If the selected item in the **Project** window isn't a component, **SelectedVBComponent** returns **Nothing**.

Note All objects in this object model have a **VBE** property that points to the **VBE** object.

Visual Basic Extensibility Reference

Visual Studio 6.0

VBForm Object

See Also Example Properties Methods Events

Returns a component in a project.

Syntax

VBForm

Remarks

The **ClassName** property determines the default value of the **VBForm** object.

The **VBForm** object replaces the **ControlTemplate** object from Visual Basic version 4.0.

Visual Basic Extensibility Reference

Visual Studio 6.0

VBNewProjects Collection

See Also Example Properties Methods Events Specifics

Represents all of the new projects in the development environment.

Remarks

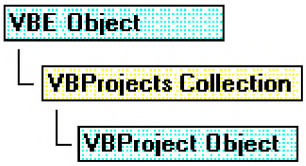
Use the **VBNewProjects** collection to access specific projects in an instance of the development environment. **VBNewProjects** is a standard collection that you can iterate through using a **For Each** block.

Visual Basic Extensibility Reference

Visual Studio 6.0

VBProject Object

See Also Example Properties Methods Events Specifics



Represents a project.

Remarks

Use the **VBProject** object to set properties for the project, to access the **VBComponents** collection, and to access the **References** collection.

Visual Basic Extensibility Reference

Visual Studio 6.0

VBProjects Collection

See Also Example Properties Methods Events Specifics

Represents all the projects that are open in the development environment.

Remarks

Use the **VBProjects** collection to access specific projects in an instance of the development environment. **VBProjects** is a standard collection that can be used in a **For Each** block.

Visual Basic Extensibility Reference

Visual Studio 6.0

VBProjectsEvents Object

See Also Example Properties Methods Events

Represents a source of events that occur when projects are added, removed, renamed, or activated in a Visual Basic project.

Syntax

VBProjectsEvents

Visual Studio 6.0

Visual Basic: MSChart Control

View3D Object

See Also Example Properties Methods Events

Represents the physical orientation of a three-dimensional chart.

Syntax

View3D

Visual Basic: MSChart Control

View3D Object Example

The following example sets the chart elevation and rotation for a three-dimensional bar chart using the view object.

```
Private Sub Command1_Click()
   ' Set the chart type to 3d bar.
   Form1.MSChart1.ChartType = VtChChartType3dBar
   With Form1.MSChart1.Plot.View3d
      .Elevation = 90   ' Look directly down onto the
                        ' top of the chart.
      .Rotation = 90
   End With
End Sub
```

Visual Basic Reference

Visual Studio 6.0

HScrollBar, VScrollBar Controls

See Also Example Properties Methods Events

Scroll bars provide easy navigation through a long list of items or a large amount of information. They can also provide an analog representation of current position. You can use a scroll bar as an input device or indicator of speed or quantityfor example, to control the volume of a computer game or to view the time elapsed in a timed process.

Syntax

HScrollBar

VScrollBar

Remarks

When you're using a scroll bar as an indicator of quantity or speed or as an input device, use the **Max** and **Min** properties to set the appropriate range for the control.

To specify the amount of change to report in a scroll bar, use the **LargeChange** property for clicking in the scroll bar, and the **SmallChange** property for clicking the arrows at the ends of the scroll bar. The scroll bar's **Value** property increases or decreases by the values set for the **LargeChange** and **SmallChange** properties. You can position the scroll box at run time by setting **Value** between 0 and 32,767, inclusive.

Visual Studio 6.0

Visual Basic: MSChart Control

VtColor Object

See Also Example Properties Methods Events

Describes a drawing color in a chart.

Syntax

VtColor

Visual Basic: MSChart Control

VtColor Object, FillColor Property Example

The following example sets the fill color for a chart backdrop brush. The **FillColor** property returns a reference to the **VtColor** object.

```
Private Sub Command1_Click()
' Sets Backdrop to Fill - Brush Style.
   MSChart1.Backdrop.Fill.Style = VtFillStyleBrush
   ' Sets chart fill color to red.
   With MSChart1.Backdrop.Fill.Brush.FillColor
      .Red = 255   ' Use properties to set color.
      .Green = 0
      .Blue = 0
   End With
End Sub
```

Visual Studio 6.0

Visual Basic: MSChart Control

VtFont Object

See Also Example Properties Methods Events

The font used to display chart text.

Syntax

VtFont

Visual Basic: MSChart Control

VtFontObject Example

The following example sets the font parameters for a chart title.

```
Private Sub Command1_Click()
   ' Ask user to supply a title.
   MSChart1.Title.Text = InputBox("Title?")
   ' Make Chart Title visible.
   MSChart1.Title.Location.Visible = True
   ' Set font for Chart Title.
   With MSChart1.Title.VtFont
      .Name = "Times New Roman"
      .Size = 18
      .Style = VtFontStyleBold Or _
      VtFontStyleItalic
      ' Use both StrikeThrough and Underline in the
      ' text.
      .Effect = VtFontEffectStrikeThrough Or _
      VtFontEffectUnderline
      ' Set text color to Blue.
      .VtColor.Set 0, 0, 255
   End With
End Sub
```

Visual Studio 6.0

Visual Basic: MSChart Control

Wall Object

See Also Example Properties Methods Events

A planar area depicting the y axes on a three-dimensional chart.

Syntax

Wall

Visual Basic: MSChart Control

Wall Object Example

The following example displays a colored wall for a three-dimensional chart.

```
Private Sub Command1_Click()
   ' Displays a colored wall for a 3D chart.
   Form1.MSChart1.ChartType = VtChChartType3dBar
   With Form1.MSChart1.Plot.Wall
      .Brush.Style = VtBrushStylePattern
      .Brush.Index = VtBrushPatternChecks
      .Brush.FillColor.Set 255, 120, 120
      .Brush.PatternColor.Set 120, 120, 0
      .Width = 20
   End With
End Sub
```

Visual Basic Reference

Visual Studio 6.0

WebClass Object

See Also Example Properties Methods Events

Contains **WebItems** (typically HTML documents) that are sent to a Web browser in response to HTTP requests.

Syntax

WebClass

Remarks

A **WebClass** object resides on a Web server. You use it to intercept HTTP requests in order to process associated Visual Basic code and return an HTML document or other **WebItems** to the browser.

Visual Basic Reference

Visual Studio 6.0

WebClassError Object

See Also Example Properties Methods Events

When processing a FatalErrorResponse event, this object indicates what error occurred.

Syntax

WebClassError

You can access this object directly from the **WebClass** object as:

```
WebClass.Error
```

Or more simply:

```
Error
```

Visual Basic Reference

Visual Studio 6.0

WebItem Object

See Also Example Properties Methods Events

Represents an HTML template or custom WebItem attached to a **WebClass** object. A **WebItem** processes and/or responds to an HTTP request.

Syntax

WebItem

Remarks

You can access a **WebItem** object directly from the **WebClass** object using:

```
Webclass.WebItem1
```

Or, more simply:

```
WebItem1
```

An HTML template is a **WebItem** that references an HTML file. This is returned as the HTTP response. A custom **WebItem** contains user-written code in its **Response** method that creates the HTTP response.

Visual Basic Reference

Visual Studio 6.0

WebItemProperties Object

See Also Example Properties Methods Events

Contains a collection of user-defined properties associated with a **WebItem** object.

Syntax

WebItemProperties

Remarks

You define the properties in this collection. The **WebItemProperties** object is a property of a WebItem. It is a collection of Variants which can be used to organize state within a wcRetainInstance WebClass, on a per-WebItem basis.

Example

```
Dim UserName As String
WebItem1.Properties("User Name") = "Someone"
```

Visual Studio 6.0

Visual Basic: MSChart Control

Weighting Object

See Also Example Properties Methods Events

Represents the size of a pie in relation to other pies in the same chart.

Syntax

Weighting

Visual Basic: MSChart Control

Weighting Object Example

The following example shows the weighting of a pie chart.

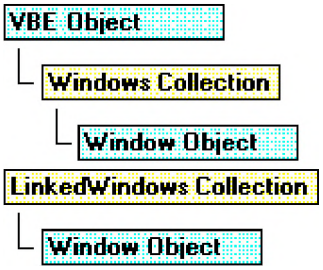
```
Private Sub Command1_Click()
   ' Show the weighting of the pie.
   Form1.MSChart1.ChartType = VtChChartType2dPie
   With Form1.MSChart1.Plot.Weighting
      .Basis = VtChPieWeightBasisTotal
      .Style = VtChPieWeightStyleArea
   End With
End Sub
```

Visual Basic Extensibility Reference

Visual Studio 6.0

Window Object

See Also Example Properties Methods Events Specifics



Represents a window in the development environment.

Remarks

Use the **Window** object to show, hide, or position windows.

You can use the **Close** method to close a window in the **Windows** collection. The **Close** method affects different types of windows as follows:

Window	**Result of using Close method**
Code window	Removes the window from the **Windows** collection.
Designer	Removes the window from the **Windows** collection.
Window objects of type linked window frame	Windows become unlinked separate windows.

Note Using the **Close** method with code windows and designers actually closes the window. Setting the **Visible** property to **False** hides the window but doesn't close the window. Using the **Close** method with development environment windows, such as the Project window or Properties window, is the same as setting the **Visible** property to **False**.

You can use the **SetFocus** method to move the focus to a window.

You can use the **Visible** property to return or set the visibility of a window.

To find out what type of window you are working with, you can use the **Type** property. If you have more than one window of a type, for example, multiple designers, you can use the **Caption** property to determine the window you're working with. You can also find the window you want to work with using the **DesignerWindow** property of the **VBComponent** object or the **Window** property of the **CodePane** object.

Visual Basic Extensibility Reference

Visual Studio 6.0

Windows Collection

See Also Example Properties Methods Events Specifics

Contains all open or permanent windows.

Remarks

Use the **Windows** collection to access **Window** objects.

The **Windows** collection has a fixed set of windows that are always available in the collection, such as the Project window, the Properties window, and a set of windows that represent all open code windows and designer windows. Opening a code or designer window adds a new member to the **Windows** collection. Closing a code or designer window removes a member from the **Windows** collection. Closing a permanent development environment window doesn't remove the corresponding object from this collection, but results in the window not being visible.

Visual Basic: Winsock Control

Visual Studio 6.0

Winsock Control

See Also Example Properties Methods Events

The **Winsock** control, invisible to the user, provides easy access to TCP and UDP network services. It can be used by Microsoft Access, Visual Basic, Visual C++, or Visual FoxPro developers. To write client or server applications you do not need to understand the details of TCP or to call low level Winsock APIs. By setting properties and invoking methods of the control, you can easily connect to a remote machine and exchange data in both directions.

TCP Basics

The Transfer Control Protocol allows you to create and maintain a connection to a remote computer. Using the connection, both computers can stream data between themselves.

If you are creating a client application, you must know the server computer's name or IP address (**RemoteHost** property), as well as the port (**RemotePort** property) on which it will be "listening." Then invoke the **Connect** method.

If you are creating a server application, set a port (**LocalPort** property) on which to listen, and invoke the **Listen** method. When the client computer requests a connection, the ConnectionRequest event will occur. To complete the connection, invoke the **Accept** method within the ConnectionRequest event.

Once a connection has been made, either computer can send and receive data. To send data, invoke the **SendData** method. Whenever data is received, the DataArrival event occurs. Invoke the **GetData** method within the DataArrival event to retrieve the data.

UDP Basics

The User Datagram Protocol (UDP) is a connectionless protocol. Unlike TCP operations, computers do not establish a connection. Also, a UDP application can be either a client or a server.

To transmit data, first set the client computer's **LocalPort** property. The server computer then needs only to set the **RemoteHost** to the Internet address of the client computer, and the **RemotePort** property to the same port as the client computer's **LocalPort** property, and invoke the **SendData** method to begin sending messages. The client computer then uses the **GetData** method within the DataArrival event to retrieve the sent messages.